Dynamic Service Composition

Judith E. Y. Rossebø
May 21, 2008
Outline

• Services and service composition
• Motivation – why dynamic service composition?
• Background for the approach
• Overview of the policy-based approach
• Example – Voice over IP service
• Conclusions
Services and Service Composition
What is a service?

A service is: an identified functionality aiming to establish some goals/effects among collaborating entities.

Captures:
- active services
- passive services
- end user services
- component interfaces (Web Services, CORBA, JINI, ...)
- layered functionality (ISO OSI)
Service essentials:

• Service is *functionality*;
  • it is behaviour performed by entities.

• Service imply *collaboration*;
  • it makes no sense to talk about service unless at least two entities collaborate.

• Service behaviour is *cross-cutting*;
  • it may depend on the behaviour of other services
  • it may be depend on sharing of resources with other services

• Service behaviour is *partial*;
  • it is to be composed with other services
Service Composition

- Service is considered as a functionality involving a collaboration among entities to achieve desired goals/effects for end users or other entities.
- Service Composition - allows for incremental service development.
  - Service components statically or dynamically combined at run time
  - Allows for reuse of components
- Modeling service composition using UML 2.x
  - UML 2.x collaborations
    - Structured way to define services in terms of collaborating roles
    - Means to decompose/compose services using collaboration uses
  - UML 2.x sequence diagrams
    - To specify service behaviour
Service Composition - dependencies

- Cross-cutting nature of services:
  - Service components interact for the execution of services
    - Services depend on each other (vertical composition)
    - Service depend on shared resources and enablers (horizontal composition)
Our approach to dynamic composition

- **Policy combined with modelling techniques**
  - To make the overall coordination of collaborations participating in a service more flexible and adaptable.

- **We focus on:**
  
  Specification of mechanisms at *service model* layer

Finally, Enable dynamic composition in the runtime system

Enable transformation to design models
Policy-based Dynamic Service Composition

Service models

Model transformation

Design models

Model transformation

Realization

dynamic composition

PESM

Policy Rules

UML Collaborations for services, connectors

Policy Rule ID | Condition | Trigger | Action | Goal
--- | --- | --- | --- | ---
PDP1_cu1 | P1, A, B, y | 11 | Do CU1 | A, g1, B, g1
PDP2_cu1 | P2, A, C, r | 12 | Do CU2 | A, g2, C, g2
PDP2_cu2 | P3, B, C, g | 13 | Do CU3 | B, g3, C, g3

Local PESMs

Local Policy

Goal | Action | Trigger | Condition | Policy Rule ID
--- | --- | --- | --- | ---
PDP1 Ar | PDP1 Ar | x | A, y | g1
PDP1 Ar | PDP1 Ar | 0 | B, C | g2
PDP1 Ar | PDP1 Ar | 0 | C, D | g3

Systems

Components, connectors

Monitoring

Service platforms

Enablers

NGN
Why Dynamic Service Compositon?
Motivation

- Securing availability of services is challenging
  - The telecommunications environment has evolved
    - from centralized to distributed
  - Services are being developed in a distributed manner in a connectionless environment requiring cooperation of several components and actors
- Example: Ensuring availability of the VoIP services.
  - a user expects to be able to place a call, complete the call without being cut off.
  - The service can be secured by e.g., providing stronger authentication
    - Can this be deployed dynamically?
Motivation

- Our approach: Development of (re-usable, flexible) patterns to ensure availability in composition
  - It should be possible to dynamically adapt, change or exchange services involved in a service collaboration
  - Example: Authentication to VoIP
    - Our approach should support exchanging weaker SIP authentication with stronger IMS AKA authentication...

```
User
authenticatee
authenticatee
UAs1:UTPA HTTP Digest
UAs1:MTPA IMS AKA
Service
authenticatee
authenticatee

Extension of SIP authentication providing mutual authentication (USIM symmetric key-based)
Using IMS AKA
```
Background
Collaborations and roles

- **An elementary collaboration** is a service collaboration with exactly two elementary roles that collaborate on an interface.
- **An elementary role** is the smallest modelling element.
- **Composite roles** are compositions of elementary roles.
  - In dynamic composition, these are dynamic compositions of elementary roles.
Composing AA-Patterns and Services:

- We provide a framework and classification of authentication and authorization patterns
  - For composing with services
  - To ensure that services are accessible to the authorized users only.
Classification of authentication patterns

- TwoParty Authenticate
  - Unilateral Authenticate
    - UniOnePass Authenticate
    - UniTwoPass Authenticate
  - Mutual Authenticate
    - MTwoPass Authenticate
    - MThreePass Authenticate
Classification of some unilateral two pass authentication patterns

UniTwoPass Authenticate

UniTwoPass Authenticate Symmetric

ISO/IEC 9798 -2 5.1.2 Two pass authentication

UniTwoPass Authenticate crypto check function

ISO/IEC 9798 -4 Two Pass

UniTwoPass Authenticate Asymmetric

ISO/IEC 9798 -2 5.1.2 Two pass authentication

UniTwoPass Authenticate using a Hash Function

rfc 2617 HTTP digest with MD 5
Specifying AA-patterns:

- **UML 2.x collaboration diagram for generic two party AA-patterns**
  - Specializations of generic patterns with properties/constraints and goals
- **UML 2.x sequence diagrams for modelling behavior**
  - Interactions uses - for flexibility and reusability
    - *Subject to service constraints (e.g. timing, processor capacity available, strength of algorithm required)*
Composition of VoIP and User Pull patterns

User

SecureVoIP (Userpull)

AccessControl Server

authenticatee → authenticator

UAs1: Authentication

UAs2: Activation

voip_req_or

voip_req_or

auths_req_or

auths_granter

USs1: Request VoIP

USs2: CAR

USs3: VoIP Use

USs4: END VoIP

authorisee

authorisor

voip_user

voip_provider

rel_er

rel_ee

voip_req_ee

t

voip_user

voip_provider
Policy and service management

A policy rule is a rule that defines a choice in the (course of the) behavior of a system [Damianou, Dulay, Lupu, and Sloman]

- Applied to dynamic service composition:
  - Policy rules are specified separately from the roles and collaborations
  - Enables collaborating roles to be dynamically composed at runtime
  - Allows for changes to be made at runtime by changing policy rules
    - Rather than changing the specifications of the roles themselves
Why apply policy?

- To separate the ordering of the roles from the roles themselves
  - To facilitate reusibility of roles in service composition
- Enable dynamic composition at runtime
  - Policy is used to govern the composition of roles executed by actors dynamically
- Making e.g. access control policies more evident in service models
- Make it possible to connect the service specification to higher level strategies and goals
  - E.g. level of security to be provided, context parameters
  - To assist in selecting the most appropriate AA-patterns to apply.
What is a policy?

- A policy is a set of policy rules.
- A policy **rule is** of the following form: if *trigger* T and *condition* C then *action* A and *goal* G
  - The **trigger** part (an event) describes when the policy should be applied
    - A message or signal associated with the collaboration or
    - An external signal or message from some other part of the system
  - The **condition** part defines constraints on its applicability
  - The **action** part defines what is to be done
    - For composition policies this means execute the collaboration
  - The **goal** part defines what is the desired result when the policy is applied.

PDP1_rule2:
T: ? (Request authentication, External, User)
C: Rel (User.secret, AcS.knowledge), User.True,
   AcS.(Unilateral required AND Challenge required)
A: UAs1b (authenticateeT -> User, authenticatorT -> AcS) : UTPA
G: User.unilaterally_authenticated, AcS.authentication_successful
Overview of the approach
Overview of the approach

- **Service models**
  - Service models
  - Design models
  - Runtime system
Overview of the policy-driven approach (1/3)

- **Service models**
  - Collaboration diagrams composed from elementary collaborations
  - Choreography is defined by the PESM
    - \textit{PDPi} - policy rules govern behavior associated with the collaborations (dependencies between sub-collaborations)
  - Policy rules stored in table with a pointer to the PDP the rule applies to

### Service models

<table>
<thead>
<tr>
<th>Policy Rule ID</th>
<th>Condition</th>
<th>Trigger</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDP1-rule1</td>
<td>P, A.a, B.b,</td>
<td>?(m,E,A)</td>
<td>CU</td>
</tr>
<tr>
<td>PDP3-rule1</td>
<td>T, B.k, C.i</td>
<td>?(m2,B,C)</td>
<td>CU3</td>
</tr>
<tr>
<td>PDP3-rule2</td>
<td>T, A.r, C.s</td>
<td>?(m3,A,C)</td>
<td>CU4</td>
</tr>
</tbody>
</table>

Trigger T: A receives message \( m \) from E (external)
Overview of the approach

- Service models
- Design models
- Runtime system
Overview of the policy-driven approach (2/3)

- **Design models**
  - Define design components in the form of statemachines that can perform the roles defined in the service models
  - Based on global PESM diagram and global composition policy
    - *Local PESM is derived for each composite role*
    - *Policy rules are distributed, (e.g. table for each local pesm)*

```
<table>
<thead>
<tr>
<th>Policy Rule ID</th>
<th>Condition</th>
<th>Trigger</th>
<th>Action</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDP2_C_rule1</td>
<td>True</td>
<td>?RoleReq  (r2)</td>
<td>PlayRole (r2)</td>
<td>r2.goal</td>
</tr>
</tbody>
</table>
```

Policy Rules
Overview of the approach

- Service models

- Design models

- Runtime system
  - Dynamic composition
Overview of the policy-driven approach (3/3)

- **Runtime system**
  - Service parts/components interact for the execution of services
  - Design models are transformed to implementations that can be deployed in the runtime system
    - *Bound to system parts depending on role-binding policies*
    - *Policy is distributed to the parts of the system*
    - *Policy enforcement agents in the system parts ensure that policy is enforced*
  - Dynamic composition (policy-based)
Example: VoIP service
Basic VoIP service

- Composed of three elementary collaborations
- The User composite roles plays three elementary roles:
  - voip_req_or
  - voip_user
  - vel_er
Composition of VoIP and User Pull patterns

User

Service Provider

Authentication

Secure VoIP (Userpull)

Access Control Server

UAs1: Authentication

auths_granter

auths_req_or

UAs2: Auths Activation

USs1: Request VoIP

voip_req_or

voip_user

USs2: CAR

authorisee

authorisor

voip_provider

USs3: VoIP Use

rel_er

rel_e

USs4: END VoIP

voip_user

voip_req_ee

voip_provider

voip_user
Global PESM:

- **Policy governed**
  - composition of two party collaborations
  - CHOICE of Authentication

PDP1_rule1:
T: ? (Request authentication, External, User)
C: Rel (User.secret, AcS.knowledge), User.True,
    AcS. (Unilateral required AND Challenge NOT required
A: UAs1a (authenticateeO -> User, authenticatorO -> AcS) : UOPA
G: User.unilaterally_authenticated, AcS.authentication_successful

PDP1_rule2:
T: ? (Request authentication, External, User)
C: Rel (User.secret, AcS.knowledge), User.True,
    AcS. (Unilateral required AND Challenge required)
A: UAs1b (authenticateeT -> User, authenticatorT -> AcS) : UTPA
G: User.unilaterally_authenticated, AcS.authentication_successful

PDP1_rule3:
T: ? (Request authentication, External, User)
C: Rel (User.secretA, AcS.secretB), User.True, AcS.Mutual required
A: UAs1c (authenticateeA -> User, authenticateeB -> AcS) : MTPA

SARDAS 2008 / Judith Rosseba, Telenor and NTNU
Global PESM - transformation to local PESM

- **PDP1**
  - UAs1a (authenticateeO->User, authenticatorO->AcS) : UOPA
  - UAs1b (authenticateeT->User, authenticatorT->AcS) : UTPA
  - UAs1c (authenticateeA->User, authenticateeB->AcS) : MTPA

- **PDP2**
  - UAs2 (authsrequestor -> User, authsgranter->AcS) : AA

- **PDP3**
Local PESM - Access Server Composite role:

PDP1_AcS_rule1:
T: ?(RoleReq(authenticatorO), User, AcS)
C: Unilateral required
A: PlayRole( authenticatorO)
G: authentication_successful

PDP1_AcS_rule2:
T: ?(RoleReq(authenticatorT) ,User, AcS)
C: Unilateral required AND Challenge required
A: PlayRole(authenticatorT)
G: authentication_successful

PDP1_AcS_rule3:
T: ?(RoleReq(authenticateeB), User, AcS)
C: Mutual required
A: PlayRole( authenticateeB)
G: authct(User)
Conclusions
Conclusions

- We have presented an approach to using policy in dynamic service composition for coordinating separately specified (re-usable) behaviors
- **Policy enforcement state machine diagram (PESM)**
  - Diagram specifies the coordination of elementary service collaborations
  - Overall choreography becomes more flexible and adaptable
- **Policy**
  - Allows for choice between alternatives of pre-defined behavior, and ordering of these
  - Can involve variables outside of a collaboration, e.g. context or requirements on strength of security to be provided
- **Composition policies**
  - Provide necessary information for linking the separately specified behaviors together
  - Provide policy enforcement information for enabling and governing dynamic service composition
Thank you for listening

Questions?